

Design Tip

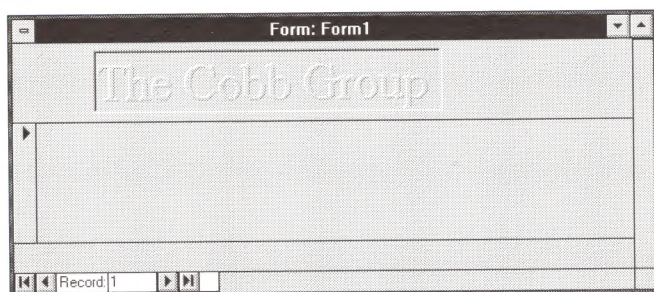
Embossed text gives your forms a classic appearance

Perhaps one of the most eye-catching effects you can create on an Access form is embossed and recessed control bor-

ders. The regions the borders enclose seem to rise up from or sink below the form's surface. You can create an embossed effect for a control's border by simply selecting the control and clicking the Raised or Sunken option button on the palette.

Unfortunately, Access doesn't provide a formatting option to create the embossed or recessed effect for text, which can be a nice design touch for form titles and other labels. For instance, wouldn't you like to format your form's title as we've done in Figure A? In this article, we'll show you how.

Figure A



We formatted this form's title by using our embossing technique.

IN THIS ISSUE

- Embossed text gives your forms a classic appearance 1
- Combining text with field entries in form and report controls 4
- Managing the billing and shipping addresses on an order form 5
- Saving money on postage by printing Postnet bar codes on envelopes 9
- Encoding ZIP+4 and Delivery Point codes in the Postnet system 12
- Using query fields to generate other query fields 13
- A fast way to delete Import Errors tables 14
- Be sure to execute SetWarnings True at the end of your Access Basic functions 14

The technique

Embossed text usually remains the same color as the form's background. The special effect results from an imaginary "light source" shining across the text from the upper-left corner. The light source seems to highlight one side of the text and cast a shadow on the other.

You can produce this effect on Access forms by using three overlapping label controls—each displaying the same text but using different colors. One label shows the text in the same color as the background, and the other labels display the text in white and dark gray. The white and dark-gray labels reside behind the first label, offset slightly so that the edges of the letters peek out. When you position and format the letters in this way, the white label mimics the highlight on one side, and the dark-gray label creates the shadow effect.

To illustrate this technique, we'll re-create the form header shown in Figure A. However, keep in mind that you can apply these steps to any label.

An example

Let's create a new form and format the title in the form's Header section with embossed text. Click the Form button in the Database window and then click New. In the New Form dialog box, click the Blank Form button.

When the new Form window appears, set the background color of the form. Click in

an empty area and then open the palette by clicking the Palette button (☞) on the tool bar. Next, click the light-gray box on the Fill row. Then, create the form's header and footer sections by pulling down the Layout menu and selecting the Form Hdr/Ftr option.

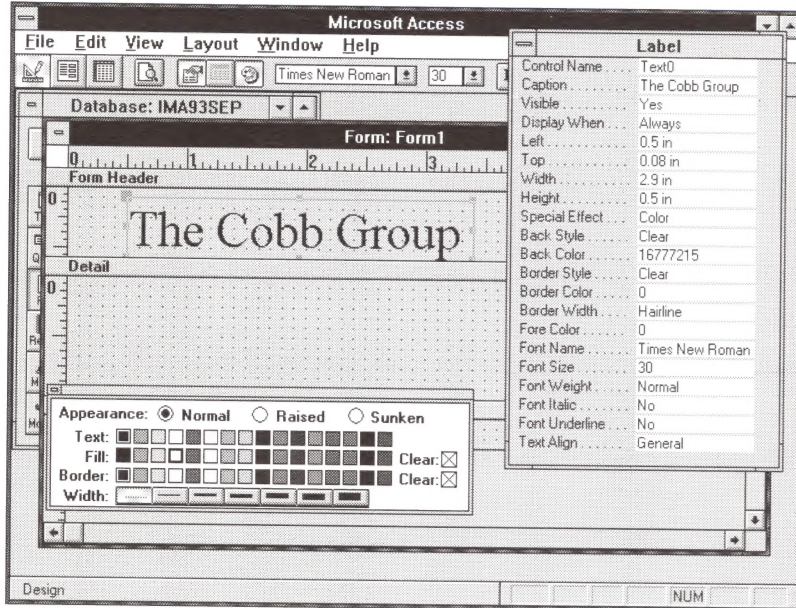
Now, place the label you want to format as embossed text. Select the label tool (A) and place a new label control in the Form Header section. After you place the label, the insertion point cursor will reside in the control. Type the text *The Cobb Group* and then press [Enter].

In a moment, you'll duplicate this label to create the three components that together produce the embossed effect. But first, we'll assign several properties that all three components will share.

First, return to the palette and click the Clear box on the Fill row. Then, open the property sheet by clicking the Properties button (☞) on the tool bar and set the Font Name and Font Size properties. For our example, we've set Font Name to *Times New Roman* and Font Size to 30, as shown in Figure B. You may need to resize the label control to fit the newly sized text.

Before duplicating the label, you should do a little preliminary work. First, write down the values in the Top and Left properties. As we mentioned, you'll slightly offset the copies of the label from the original

Figure B



We'll format this label to create an embossed effect.

Inside MICROSOFT ACCESS™

Inside Microsoft Access (ISSN 1067-8204) is published monthly by The Cobb Group.

Prices: Domestic \$59/yr. (\$7.00 each)
Outside US \$79/yr. (\$8.50 each)

Phone: Toll free (800) 223-8720
Local (502) 491-1900
Customer Relations Fax (502) 491-8050
Editorial Department Fax (502) 491-4200

Address: You may address tips, special requests, and other correspondence to

The Editor, *Inside Microsoft Access*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster: Second class postage is pending in Louisville, KY. Send address changes to

Inside Microsoft Access
P.O. Box 35160
Louisville, KY 40232

Copyright:

Copyright © 1993, The Cobb Group. All rights reserved. *Inside Microsoft Access* is an independently produced publication of The Cobb Group. The Cobb Group reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for both personal and commercial use.

The Cobb Group, its logo, and the Satisfaction Guaranteed statement and seal are registered trademarks of The Cobb Group. *Inside Microsoft Access* is a trademark of The Cobb Group. Paradox is a registered trademark of Borland International. dBASE III and dBASE III PLUS are registered trademarks of Ashton-Tate, a Borland International company. Microsoft and MS-DOS are registered trademarks of Microsoft Corporation. Microsoft Windows and Word for Windows are trademarks of Microsoft Corporation.

Staff:

Editor-in-Chief David Brown
Editing Timothy E. Hampton
Elizabeth Welch
Production Artist Julie Jefferson
Design Karl Feige
Publications Manager Tara Dickerson
Managing Editor Suzanne Thornberry
Circulation Manager Brent Shean
Publications Director Linda Baughman
Editorial Director Jeff Yocom
Publishers Mark Crane
Jon Pyles

Back Issues:

To order back issues, call Customer Relations at (800) 223-8720. Back issues cost \$7 each, \$8.50 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you. Please identify the issue you want by the month and year it was published. Customer Relations can also provide you with an issue-by-issue listing of all articles that have appeared in *Inside Microsoft Access*.

label's position, which the Top and Left properties store. In our example, we've placed the form so that Top equals 0.08 inches and Left equals 0.50 inches.

Also, move to the Control Name property and enter *Title*. As you'll see, naming the labels helps you distinguish among the three label controls necessary to produce the effect.

Creating the embossed effect

Now you're ready to create the embossed effect for the label. You'll first create a copy of the label that will act as the embossed highlight. You'll then create another copy that will serve as the shadow.

With the label control selected, pull down the Edit menu and click the Duplicate command. Then, switch to the property sheet and assign *Title* - to the Control Name property. Also, decrease the Left and Top properties by 0.01. In our example, Left becomes 0.49, and Top becomes 0.07. Next, return to the palette and click the white box in the Text row. Finally, select the Send To Back command from the Layout menu so that the highlighted version of the label hides behind the Title label. Figure C shows our form at this point.

Now, you make another copy of the Title label. Press [Shift][Tab] to return to the original Title label control. Then, select the Duplicate command from the Edit menu again. Next, switch to the property sheet for this control and change the Control Name property to *Title* + and increase the Left and Top properties. In our example, Left will equal 0.51, and Top will be 0.09.

Once you've positioned the label, return to the palette and click the dark-gray box on the Text row. Finally, pull down the Layout menu and select the Send To Back command so that the Title Shadow label also hides behind the Title label.

Finally, you must change the color of the original Title label text to match the form's background color, light gray. Return to the Form window by clicking the window's title bar and pressing [Shift][Tab]. Next, move to the palette and click the light-gray box on the Text row. Figure D shows the result.


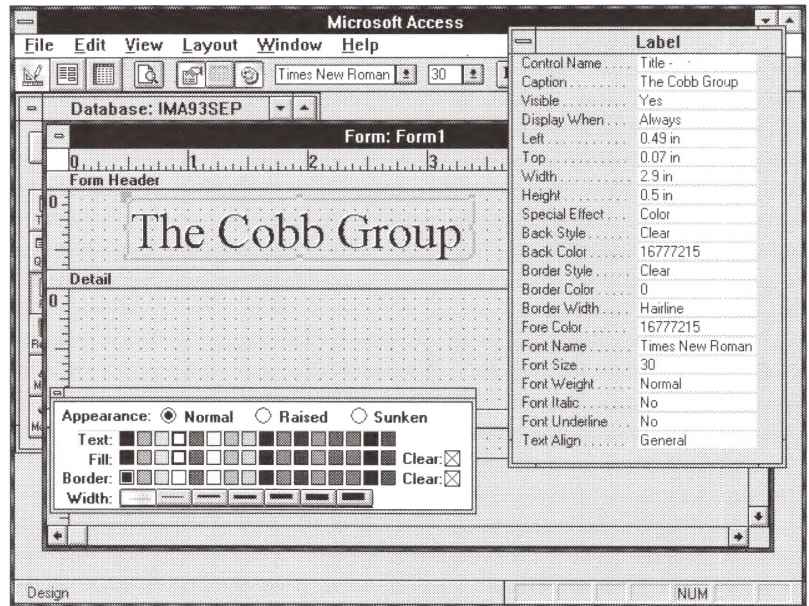
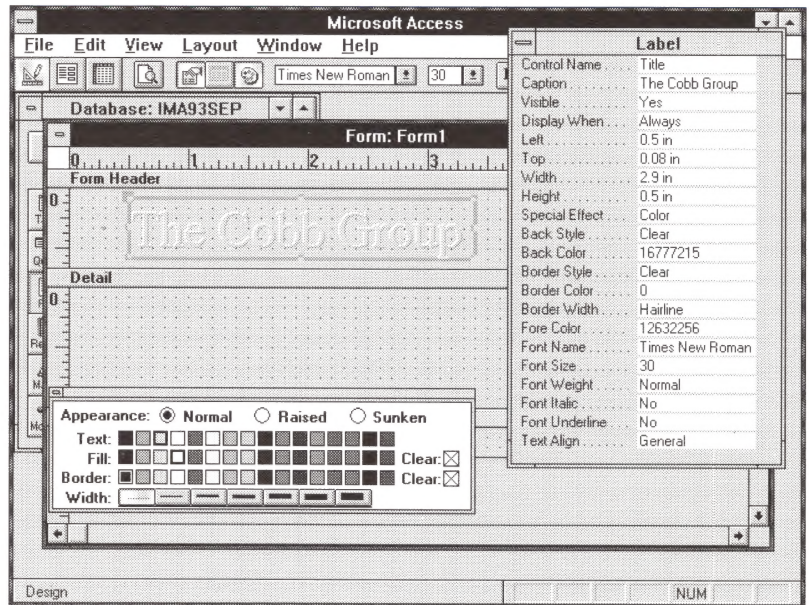
When you click the Form View button  on the tool bar, you'll see the embossed title shown in Figure E on page 4. To achieve the recessed effect for the outer rectangle shown in Figure A on page 1, you select the

Figure C



The label you offset above and to the left of the first label will apply the highlight.

Figure D



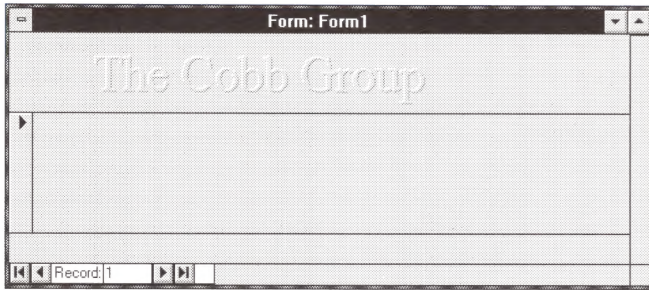
When you finish placing and formatting the label controls, the form title should look like this.

Title label, move to the palette, and click the Sunken option button.

Creating recessed text

You use the same technique when you want a recessed effect for your label text. However, you reverse the colors of the labels that offset the primary label. In our example, you assign the dark-gray color to the Title - label and the white color to the Title + label.

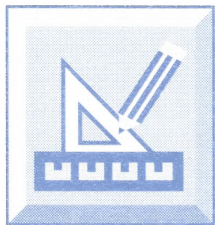
Figure E



Here's the form title in Form view.

Conclusion

In this article, we showed you how to format label text with the same embossed effect Access automatically provides for label borders. You create this effect by overlapping three copies of the same label and manipulating the color and positions of those labels. The result is an interesting effect you can use to spice up almost any form. You can use the same technique to create recessed text as well. ♦



Design Tip

Combining text with field entries in form and report controls

In "Saving Money on Postage by Printing Postnet Bar Codes on Envelopes," on page 9, we explain how you can combine a ZIP Code field with other text values by using the concatenation operator, &. Although in that particular situation the concatenation operator offers the best way to combine the elements of the expressions, there's another—often much better—way to do the same thing.

In a nutshell, you begin the expression with a quotation mark, just as you would an ordinary text entry. Then, you can embed a field within the entry by typing the field identifier enclosed in pipe characters (|). After the closing pipe character, the text entry can resume. Of course, you conclude the entire expression with a closing quotation mark.

Let's look at the simple example that appears on page 11. This technique frames the ZIP Code field entries with the frame bars provided by the S character. We used

```
= "S" & [ZIP Code] & "S"
```

which uses concatenation operators to piece together the components of the field. However, we could have used the expression

```
= "S|[ZIP Code]|S"
```

to embed the ZIP Code field in the string.

Combining name data

Let's look at another example that arises in almost every Access user's reports. You often

want to display First Name and Last Name field entries in one text box control, separating them with a space.

Before you create this control, you must first create a new report for the Customers table that you created for the other article. Highlight the Customers table in the Database window and click the New Report button (📄) on the tool bar. Then, in the New Report dialog box, click the BlankReport button.

When the blank report appears, create an unbound text box control by clicking the Text Box tool (📄) in the Toolbox and then clicking in the Detail section of the Report window. Next, click again within the control's borders to obtain the insertion point cursor. As you probably know, when you type inside the control, you set its Control Source property.

If you use the concatenation operator to combine First Name and Last Name fields, you use the expression

```
= [First Name] & " " & [Last Name]
```

On the other hand, when you use the pipe character to embed the fields in a text entry, you enter the expression

```
= "[First Name] | [Last Name]"
```

The only text in this example is the space character between the fields. When you use the pipe-character technique, the expression is more readable and simply easier to type. ♦

Managing the billing and shipping addresses on an order form



Macro Tip

Most people who buy Microsoft Access do so to organize their business' data processing needs, which usually translates into an order-tracking system for the products they sell. Two important elements of such a database are the billing address and the shipping address. You keep these types of addresses separate because bills usually go to company headquarters, while shipments often go to various sites.

As a matter of fact, the two types of addresses usually reside in separate tables. The billing address often resides in the Customers table because a customer's billing address rarely changes from order to order. On the other hand, a customer could request that you send a shipment anywhere. Depending on the customer, the shipping address can vary with every order.

Although it's necessary, keeping separate billing and shipping addresses can often be annoying. For instance, when the addresses are the same, you shouldn't need to type the information in the shipping-address fields when it already exists in the Customers table's billing-address fields. It would be nice to have a button on your order-entry form that copied the address information for you.

In this article, we'll show you how to add this feature to your order-entry form. Along the way, you'll learn the general technique of copying data between tables by using a macro. Let's first discuss the basic technique. Then, we'll flesh out the technique with a detailed example.

The technique

The key to this technique is a macro that copies the billing-address information from the Customers table to the Orders table's shipping-address fields. You attach the macro to a button on the order form so that you simply click the button when you want to duplicate the address information.

In developing such a macro, you face a couple of tricky obstacles. First, not only must the macro access the Customers table, but it must look up a particular customer's record. Then, it must grab the data it wants to copy from the fields in the Customers table.

Fortunately, the macro can accomplish both requirements as long as it can open a form that displays the Customers table's data. The OpenForm macro action has an argument called Where Condition, which lets you define the records that the form will display. You can set this argument so that only the data for the customer placing the order appears in the form.

Also, you can easily access data in the individual fields when you use a form. The data rests in controls on the form, which you can access by providing the controls' identifiers.


If you haven't attempted much macro designing, this may seem a little intimidating to you. Don't worry—these concepts will become clearer after you work through an example. In the following pages, we'll set up sample tables and forms and help you build the macro step by step.

Building your order-entry database

Although there are many ways to incorporate customer information in your order-entry database, almost all include separate tables for storing customer information, order data, and product descriptions. However, in this example, we'll concentrate on only the tables that store customer and order information. Table A shows the structure of the Customers table we'll use in our examples, and Table B on page 6 shows the structure of the Orders table. Figure A, also on page 6, shows some sample data for the tables.

Notice how these tables store the address information. The Customers table holds in-

Table A

Customers Table			
Key	Field Name	Data Type	Field Properties
	Customer ID	Counter	
	Company Name	Text	Field Size =
	Billing Address	Text	Field Size = 40
	Billing City	Text	Field Size = 15
	Billing State	Text	Field Size = 2
	Billing ZIP Code	Text	Field Size = 10

formation that stays the same for every order, such as the customer name and billing address. The Orders table stores data that's specific to each order, such as the purchase date, the products ordered, and the shipping address. At this point, the shipping-address fields are blank so you can later use the but-

ton to copy the Customers table's billing-address data.

Now that we've defined the Customers and Orders tables, let's create forms for the tables to use in our examples. To simplify the example, we'll use the Single-column form wizard to create forms with the most basic layout.

Table B

Orders Table			
Key	Field Name	Data Type	Field Properties
	Invoice ID	Counter	
	Customer ID	Number	Field Size = Long Integer
	Shipping Address	Text	Field Size = 40
	Shipping City	Text	Field Size = 15
	Shipping State	Text	Field Size = 2
	Shipping ZIP Code	Text	Field Size = 10
	Purchase Date	Date/Time	
	Order Amount	Currency	

Creating the order form

First, let's create the order form. Highlight the Orders table in the Database window and click the New Form button (📄) on the tool bar. Then, in the New Form dialog box, click the Form Wizards button. In the following dialog box, select Single-column and click OK. When the first wizard dialog box appears, simply click the Fast Forward button (➤) in the lower-right corner. Then, on the final wizard dialog box, click the Design button to enter the Design view of the form.

Figure B shows the default form layout you'll see.

Next, consolidate the shipping-address fields on the right side of the report, as shown in Figure C. Start by first deleting the shipping-address fields' labels. Do so by highlighting each label and pressing [Del]. Then, arrange the text box controls individually. Finally, create the Shipping

Address: label by selecting the Label tool (A) from the Toolbox, clicking on the form where you want the text to reside, and then typing *Shipping Address:* in the control.

Now, create the button that will execute the address-copying macro. Start by selecting the command button tool (📄) from the Toolbox. Next, click below the shipping-address text boxes and then drag the text box to the right and down in order to make a button as long as the shipping-address text boxes. After the button appears, click within the button control's boundaries so that Access offers the insertion point cursor. When you have the cursor, remove the default button label, type *Duplicate Billing Address*, and press [Enter]. Figure D shows the completed order-entry form.

Figure A

Customer ID	Company Name	Billing Address	Billing City	Billing State	Billing ZIP Code
1	Nixon, Inc.	89 Jefferson Way	Portland	OR	97201
2	Wong Enterprises	55 Grizzly Peak Rd.	Butte	MT	59801
3	Jeffers, Limited	Thompson Building	Denver	CO	80227
4	McCormick Brothers	488 Sacajawea Ct.	Santa Fe	NM	87501
5	Nagy & Company	722 Dynamite Blvd.	Kirkland	WA	98034
6	Brown & Associates	9420 Bunsen Parkway	Louisville	KY	40220

Invoice ID	Customer ID	Shipping Address	Shipping City	Shipping State	Shipping ZIP Code	Purchase Date	Order Amount
920681	3					8/18/92	\$10.00
920792	6					9/1/92	\$905.00
920797	3					10/2/92	\$20.00
*	0						\$0.00

We'll use this data in our examples.

Figure B

We'll enhance this basic form with a button that copies a customer's billing address into the shipping-address fields.

Of course, we haven't created the button's macro yet, so we must stop designing the form at this point. First, save the form by issuing the Save As... command from the File menu. Then, type *Order Entry* in the Save As dialog box and click OK.

Creating the customer form

The next step is to create the customer form that the button's macro will use to access the customer's billing-address fields. Create a new form by highlighting the Customers table in the Database window and clicking the New Form button (📄) on the tool bar. Then, in the New Form dialog box, click the Form Wizards button. In the following dialog box, select Single-column and click OK. When the first wizard dialog box appears, simply click the Fast Forward button (➤) in the lower-right corner. When the final dialog box appears, click the Design button. Figure E shows the default layout the wizard generates.

We'll use this form as is. Just issue the Save As... command from the File menu. Then, type *Customer Form* in the Save As dialog box and click OK. Finally, close the form's window.

Building the address-copying macro

Finally, we're ready to create the macro that copies a customer's billing address to the order form's shipping-address fields. Start by clicking the Macros button in the Database window and then clicking New. When the window appears, enter the macro actions and arguments listed in Table C on page 8.

As you can see, the OpenForm action opens the Customer Form you just created. The Where Condition expression limits the form to displaying only those records whose Customer ID fields match the current Customer ID field in the Order Entry form.

After opening the form, the macro executes four SetValue actions to copy the four fields of the address. The Item argument specifies the control to which you want to assign a value, and the Expression argument contains the value you want to assign. In this case, you're assigning the values in the billing-address controls in the Customer Form to the shipping-address controls in the Order Entry form.

Figure C

Arrange the address fields in a "mailing-label" format on the right side of the form.

Figure D

Your form will look like this one after you add the Duplicate Billing Address button.

Figure E


We'll use this form to obtain the billing address while filling out the Order Entry form.

When you've finished, save the new macro by issuing the File menu's Save As... command. In the Save As dialog box, type *DuplicateBillAddress* and click OK. Finally, close the window.

Table C


Actions	Action Arguments
OpenForm	Form Name = Customer Form View = Form Where Condition = [Customer ID]=Forms![Order Entry]![Customer ID] Data Mode = Read Only Window Mode = Hidden
SetValue	Item = [Shipping Address] Expression = Forms![Customer Form]![Billing Address]
SetValue	Item = [Shipping City] Expression = Forms![Customer Form]![Billing City]
SetValue	Item = [Shipping State] Expression = Forms![Customer Form]![Billing State]
SetValue	Item = [Shipping ZIP Code] Expression = Forms![Customer Form]![Billing ZIP Code]
Close	Object Type = Form Object Name = Customer Form

Now, all that remains is to assign the new macro to the On Push property of the Order Entry form's command button. First, return

to the Order Entry form's Design view. (If you closed the window, you'll need to open the form by double-right-clicking the Order Entry form name in the Database window.) Then, select the Duplicate Billing Address button and click the Properties button  on the tool bar. Next, move to the

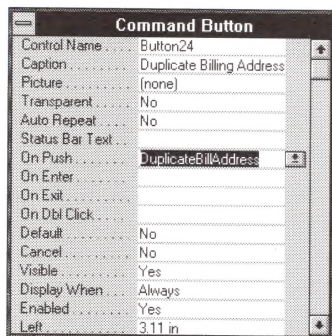
On Push property, click its drop-down arrow, and select the Duplicate-BillAddress macro from the selection list. Figure F shows the property sheet after you assign the macro to the On Push property. Then, save the form again by issuing the File menu's Save command.

Using the Order Entry form

Finally, the form is ready for use. Click the Form View button  on the tool bar. The form will initially display the first record in the Orders table. The record stores the Customer ID entry 3, which belongs to the company Jeffers, Limited. If this company places an order and wants the shipping address to be the same as the billing address, you copy that address to this form by clicking the Duplicate Billing Address button. The address will appear in the fields, as shown in Figure G.

Conclusion

In this article, we showed you how to set up your order-entry form with a feature that lets you duplicate a customer's billing address in the shipping-address fields. This feature eliminates wasted typing in those cases in which the customer wants the order shipped to the billing address. ♦

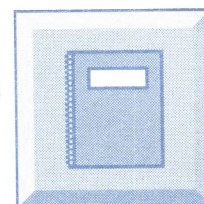
Figure F

You assign the DuplicateBillAddress macro to the On Push property.

Figure G

The customer's billing address will appear when you click the Duplicate Billing Address button.

Saving money on postage by printing Postnet bar codes on envelopes



Report Tip

Last month's "Creating Bar Code Labels in Access Reports" showed you how to create bar code labels you can stick on items in an inventory control system. This month, we'll show you how to apply the same technique to print Postnet bar codes on your mailing labels or envelopes. Essentially, you obtain a font that prints the numeric ZIP code values in the Postnet bar code format.

You can find an example of these bar codes on any letter you receive in the mail. You'll find the Postnet bar codes in the lower-right corner of the envelope or in the first line of the mailing label. If you can put the bar codes on your own mail, the post office doesn't have to do it. Consequently, they'll reduce the postage rate, provided you mail enough pieces to qualify for the discount.

The technique—a general description

Postnet bar codes are basically ZIP codes encoded with the post office's special bar-coding system. However, you encode more than just the ZIP code. You must first append the ZIP code to a correction digit, which lets the post office immediately detect read errors that occur while scanning the code. You must also enclose the encoded entry with special characters called frame bars, which tell the post office scanners where the actual code begins and ends. We'll show you how to provide those codes in a moment. First, we'll describe the PostnetBars font you'll use to print the Postnet bar codes.


The PostnetBars font


You can obtain a shareware Postnet bar code font in Library 15 in CompuServe's Access forum—just as you downloaded the font RSCode39 for last month's article. Look for the file ZIPBAR.ZIP in Library 15. Note that you must have Version 2.04g of PKUNZIP to open the ZIP file. (Remember that this font and other material included in the ZIP file are not free. You must send a fee to the developer—in this case, the fee is \$10.)

After you unzip the file, you must install the font in the Windows system. To do so, you launch the Control Panel application in

the Program Manager's Main group. Then, double-click the Fonts application. In the Fonts dialog box that appears, click the Add... button and then specify the fonts you want to add in the resulting Add Fonts dialog box.


To add a font, you first find the directory that holds the files you just extracted. Then, you highlight the PostnetBars item in the list and click OK. After you do so and then restart Windows, the font will appear in the list of fonts within all your Windows applications.

As we mentioned, the font provides a bar code segment for each digit, consisting of a combination of long and short bars (five bars total). For instance, the code for the number 4 is . To encode the ZIP code, you string together its encoded digits.

Let's now look at a Postnet bar code for an actual ZIP code—we'll use The Cobb Group's ZIP code, which is 40220. In order to create a bar code on a report, you'd create an undefined text box control and enter `= "40220"` as its Control Source property and *PostnetBars* as its Font Name property. When you do so, the bar code  will appear. We've marked the code segments for the individual digits.

Including the frame bars

The PostnetBars font also offers a bar code for the letters *s* and *S*. It provides the same bar code for both these characters—a single full-height bar—which serves as the frame bar. Remember, you must enclose the entire ZIP code, including the correction digit we mentioned, in the frame bars.

To enclose a ZIP code in frame bars in a report's text box control, you assign the ZIP code entry enclosed with *S* characters to the control's Control Source property. Then you assign *PostnetBars* to its Font Name property. For example, to enclose The Cobb Group's ZIP code with frame bars, you actually encode *S40220S*, which would appear as .

Computing the correction digit

Now let's turn to the correction digit. To calculate the digit, you sum all the digits the

ZIP code contains and then subtract the result from the next greater multiple of 10. For example, to compute the correction digit for The Cobb Group's ZIP code, 40220, you sum the digits. Then, subtract the result (8) from 10, which is the next highest multiple of 10. Since 10 minus 8 is 2, 2 is the correction digit in this example. Consequently, you'd encode the value S402202S. (Note the extra 2 as the second-to-last character.)

In principal, summing the digits of a ZIP code is easy. We can quickly pick out the digits as we scan the number and, at the same time, compute the running total. However, summing the digits in a number is a difficult thing for Access to do. Why? Well, before Access can sum the digits, it must separate them. Since Access isn't capable of doing this, you must program

the process in an Access Basic function. Your function can then compute the correction digit.

To create this function, first create a new Access Basic module by clicking the Modules button in the Database window and then clicking the New button. When the new Module window appears, pull down the Edit menu and select the New Procedure... command. Next, in the New Procedure dialog box, enter *CorrectionDigit* and click OK.

Next, enter the new function's code by typing the statements

```
Function CorrectionDigit (ZIPCode$)

    ZIPLength# = Len(ZIPCode$)
    CDigit# = 0
    For i# = 1 To ZIPLength#
        CDigit# = CDigit# + Val(Mid$(ZIPCode$, i#, 1))
    Next i#
    CorrectionDigit = 10 - (CDigit# Mod 10)

End Function
```

End Function

This function accepts the ZIP Code value you want to encode as an argument. The first statement assigns the length of the ZIP code to the *ZIPLength#* variable. The next statement initializes a new variable that's named *CDigit#*, which will hold the running sum of the ZIP code's digits. The For loop that spans the following three lines actually sums those digits by scanning through all the digits in the *ZIPCode\$* variable's value, adding each digit's contribution to *CDigit#*. Then, the last statement computes the correction digit from the running sum and assigns the result to the function name in order to return the value.

To prepare the function for use, pull down the Run command and select Compile All. Then, save the new module by selecting Save As... from the File menu, typing *Postnet Utilities* in the Save As dialog box, and clicking OK. Finally, close the module window by pressing [Ctrl][F4].


Once you've created the function, you can call it to compute a ZIP code's correction digit. For example, to create The Cobb Group's complete bar code, you'd assign to the text box's Control Source property "S40220" & CorrectionDigit("40220") & "S". The text box will print  when you use the bar code font.

Table A


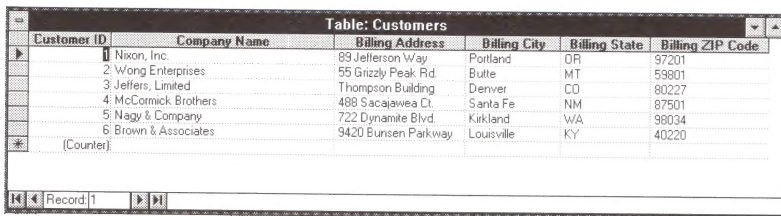
Customers Table			
Key	Field Name	Data Type	Properties
	Customer ID	Counter	
	Company Name	Text	Field Size=40
	Billing Address	Text	Field Size=40
	Billing City	Text	Field Size=15
	Billing State	Text	Field Size=2
	Billing ZIP Code	Text	Field Size=10

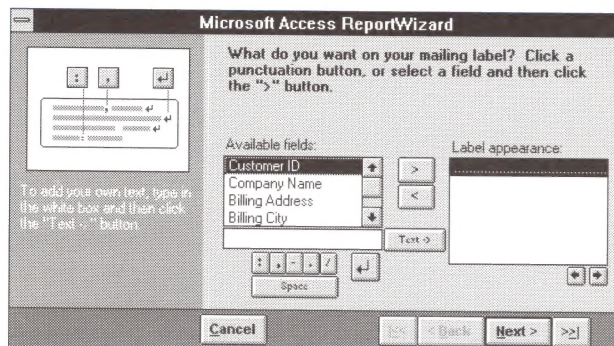
Figure A



Customer ID	Company Name	Billing Address	Billing City	Billing State	Billing ZIP Code
1	Nixon, Inc.	89 Jefferson Way	Portland	OR	97201
2	Wong Enterprises	55 Grizzly Peak Rd.	Butte	MT	59801
3	Jeffers, Limited	Thompson Building	Denver	CO	80227
4	McCormick Brothers	488 Sacajawea Ct.	Santa Fe	NM	87501
5	Nagy & Company	722 Dynamite Blvd.	Kirkland	WA	98034
6	Brown & Associates	9420 Bunsen Parkway	Louisville	KY	40220

We'll use this data in our Postnet bar code examples.

Figure B



You lay out the mailing label in the wizard's first dialog box.

Creating the bar codes on a mailing-label report

Now let's apply this technique to place bar codes on mailing labels. In our example, we'll use the Customers table, with the structure shown in Table A and the data shown in Figure A.

After you create the table and load it with data, you're ready to create the mailing-label report. To do so, highlight the Customers table in the Database window and click the New Report button (R) on the tool bar. In the New Report dialog box, click the ReportWizards button. Next, in the following dialog box, highlight the Mailing Label option and click OK. Then, define the new report by responding to the wizard dialog boxes.

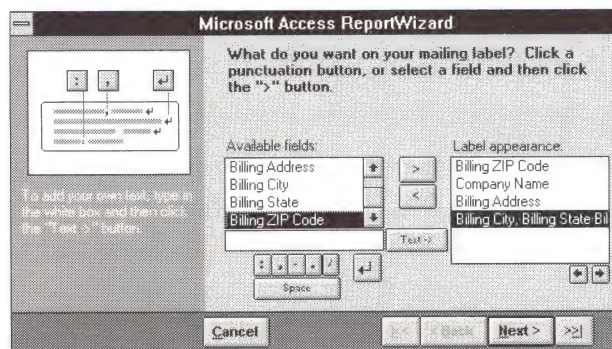
Figure B shows the first dialog box, in which you lay out the elements you want to appear on the label. As you probably know, you lay out a mailing label by selecting from the Available Fields list the fields you want to appear and separating your choices with the punctuation provided by the buttons below the list.

You start with the Billing ZIP Code field on the first line and click the Return button. You then place the Company Name field on the second line. Next, place the Billing Address field on the third line and the Billing City, Billing State, and Billing ZIP Code fields on the fourth line, separating these fields with the customary comma and spaces. Figure C shows what the first dialog box should look like at this point.

In the second dialog box, select Billing ZIP Code from the field list. That way, the wizard will generate a report that sorts by ZIP code. (To receive any discount from the post office, you must sort the mail by ZIP code.) Then, click the Next> button to continue.

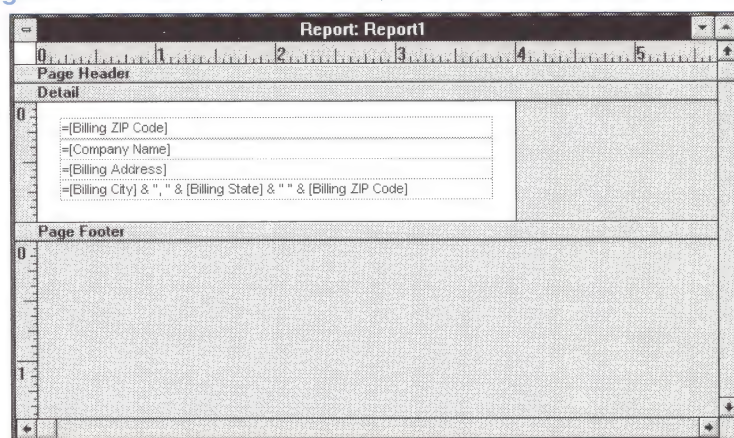
In the last dialog box, scroll down the list of Avery Numbers until you find 5161. Select that mailing label size and then click the Design button. Figure D shows the report the Mailing Label wizard generates. However,

Figure C



After you work through the steps we described, the first dialog box should look like the one shown here.

Figure D



The Mailing Label wizard generates this report.

you must make several modifications to the controls on this report before it will produce bar codes properly.

Let's start with the top control, which currently displays only the Billing ZIP Code field. You want to modify this control's Control Source so that it displays the bar codes. To do this, you first increase its height. Select the control by clicking it. Then, grab the resize handle on the top border and drag it to just below the top of the section. You should be able to almost double the control's height.

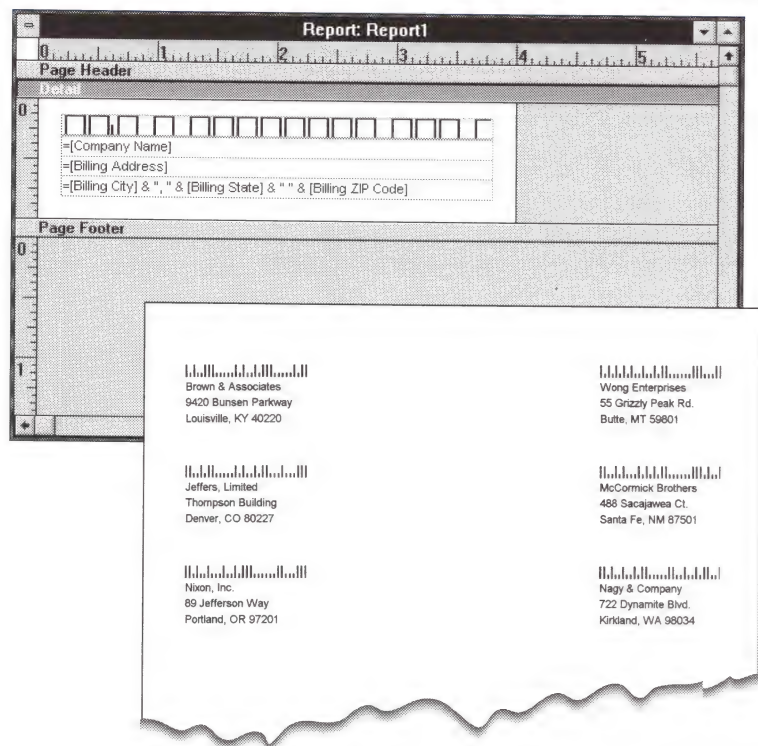
Next, change the control's Control Source property by clicking within the control borders in order to obtain the insertion point cursor. Then, change `= [Billing ZIP Code]` to

```
= "S" & [Billing ZIP Code] &  
➔ CorrectionDigit([Billing ZIP Code]) & "S"
```

which concatenates the Billing ZIP Code and correction digit and encloses the result in the frame bars.

Finally, you format the control to print in the PostnetBars font. Click the Font Name

Figure E



combo box's dropdown arrow and select PostnetBars from the list of fonts. Also, change the Font Size to 14 from the default size of 8. Figure E shows the final report and a sample printout. Before closing the window, save the report by pulling down the File menu and selecting the Save As... command. In the Save As dialog box, enter *Customer Mailing Label 5161 With Postnet* and click OK.

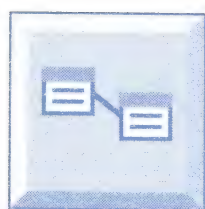
Rules and regulations

When the Postnet system first came on the scene, post offices restricted the bar codes to the bottom of the envelope. However, the Postnet system is evolving, and the equipment post offices use to read the bar codes is improving. Therefore, we suggest you contact a postal service representative in your area who can advise you further on how to implement the Postnet bar codes properly.

Conclusion

In this article, we showed you how to encode ZIP code information by using the Postnet bar code system. We also showed you how to prepare your data for placement on mailing labels. ♦

You include the Postnet bar codes at the top of the mailing label.



Query Tip

Encoding ZIP+4 and Delivery Point codes in the Postnet system

If bulk mail is an important concern for your business, you may have already investigated the post office's ZIP+4 and Delivery Point discount programs. If you provide these codes, you may be eligible for substantial discounts on bulk-rate postage.

As you may know, the ZIP+4 code includes the ZIP code and an additional four-number code to locate a destination in a ZIP code more precisely. The Delivery Point code is the full ZIP+4 code with an additional two-digit code at the end. These digits are the last two digits of the address' street number.

When you're implementing Postnet bar codes for ZIP+4 and Delivery Point codes,

you must display the entire code in the bar code font. You must also compute the correction digit for the full code. In this article, we'll guide you through this technique.

The technique

To generate Postnet bar codes for the expanded codes, you use the same tools you developed in "Saving Money on Postage by Printing Postnet Bar Codes on Envelopes," on page 9: You calculate the correction digit, enclose the code in frame bars, and assign the PostnetBars font to the control that contains code. Encoding the additional digits of the code is trivial: The font does all the

work. The only complication is computing the correction digit for the new code.

The `CorrectionDigit()` function you created will do the job as long as the code is in a proper format—namely, the code must contain only numbers. As you know, the expanded codes use a hyphen to separate the base ZIP code and the ZIP+4 information.

One solution might be to create another Access Basic function that will strip the hyphen as it sums the digits of the code. However, we suggest you avoid the additional programming in favor of a query that generates a copy of the ZIP Code field, stripping out the hyphens when it encounters them. Figure A shows a query that generates both ZIP+4 codes and Delivery Point codes.

Unfortunately, there just isn't room on the page to show the new columns' expressions in the figure. We'll show them to you one at a time.

When you're building the query, you type into the second column's Field/Expression cell

```
ZIP+4: If(InStr([ZIP Code],"-"),
➡ Left$([ZIP Code],5) &
➡ Right$([ZIP Code],4),[ZIP Code])
```

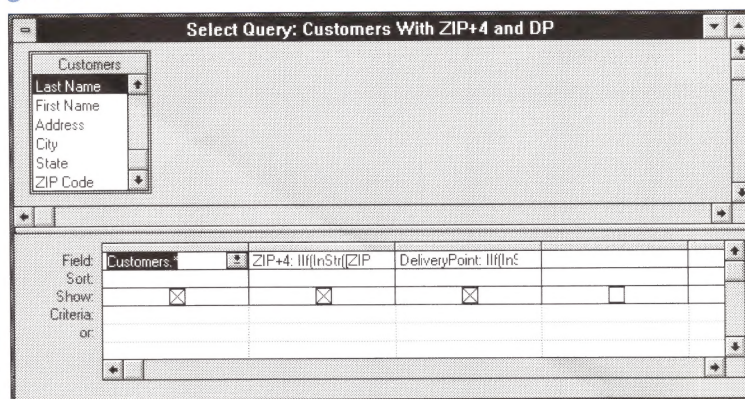
which names the new column ZIP+4 and generates the values. This expression first tests whether the ZIP Code entry contains a hyphen. If so, it assumes the hyphen separates the five-digit ZIP code and the additional four-digit ZIP+4 portion. In this case, the expression uses the `Left$()` and `Right$()` functions to grab the two pieces and then concatenates them. If a particular ZIP Code entry doesn't have a ZIP+4 extension, the query fills in only the ZIP Code field in the new columns.

To create a column for the Delivery Point code, you type

```
DeliveryPoint: If(InStr([ZIP Code],"-"),
➡ Format$([ZIP+4]) & Format$(Val([Address]) Mod
➡ 100,"00"),[ZIP Code])
```

into the third column's Field/Expression cell. This expression names the new field `DeliveryPoint` and then generates the field's value by building on the result of the ZIP+4 calculation. When you want to include Delivery Point information, you simply append

Figure A



This query converts the ZIP+4 and Delivery Point codes into a format that our `CorrectionDigit()` function can process.

the ZIP+4 code with the two rightmost digits of the address' street number. If you were wondering why the Address field crept into the expression, the answer is simple: The street number resides in the Address field.

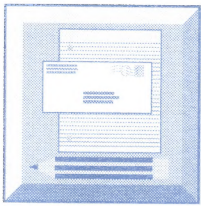
Once the query creates these fields, you can use them instead of the ZIP Code field to create the bar codes. As you may remember, you first create a text box control on the mailing label report and assign to its Control Source property the expression that encloses in 5 characters the ZIP+4 or Delivery Point code—along with its correction digit. You then assign the control the PostnetBars font at point size 14. ♦

Using query fields to generate other query fields

Often, you create a query in order to derive information from the data that resides in the table. For instance, if you store a Quantity and a Unit Cost field in an Invoice table, you can create a query that generates a Total Cost field by multiplying Quantity and Unit Cost. To do so, you'd include the expression `Total Cost: [Quantity] * [Unit Cost]` in the Field/Expression cell of the QBE grid. The text before the colon assigns the name of the new field, and the portion after the colon defines the value the query will generate.

This capability makes sense. As Access generates each row's result, it draws the information it needs for the calculation from the underlying table. But Access goes farther than that. You can also use other fields the query creates. Using the previous example, you could generate a field called Order Amount by multiplying the Total Cost field by the tax rate. You'd enter the expression `Order Amount: [Total Cost] * 1.05` into the Field/Expression cell.

We use this technique in the accompanying article to generate the query's Delivery Point field. The expression that derives the Delivery Point codes appends the query field ZIP+4 to the two street-address digits.



Letters

A fast way to delete Import Errors tables



I import a number of Lotus 1-2-3 spreadsheets every day, and with each import, Access creates another error table. I usually don't delete these tables right away; as a result, my database now contains quite a lot of them.

Now I want to get rid of them. I want an easy way to delete multiple tables at one time. I'd think that a macro could delete them faster than I can move to each table, press the [Del] key, and then click OK in the confirmation menu.

*Robin R. Barton
St. Louis, Missouri*

We can give three solutions to Ms. Barton's question. The easiest technique just lets you avoid the confirmation menu. When you highlight a table and then press [Shift][Del] instead of just [Del], Access will immediately remove the table. This method eliminates only one step from the manual technique Ms. Barton currently uses.

Although the [Shift][Del] key provides a shorter method, you still need to highlight and delete the tables one at a time. To automate this process, you must create either a macro or an Access Basic function.

Be sure to execute SetWarnings True at the end of your Access Basic functions

In the accompanying letter, we created both a macro and an Access Basic function to delete Import Errors tables. In both the macro and the function, we turned off system warning messages in order to suppress the confirmation dialog boxes that Access pops up when you try to delete a table. In the macro, we executed the SetWarnings action to do this. In the function, we issued the DoCmd command, which essentially runs the SetWarnings macro action.

In this article, we'll point out an interesting difference in the way macros and Access Basic functions implement SetWarnings. Let's look first at the macro.

The macro's second line executes the SetWarnings action, using the Warnings On = No argument, which turns off the warning messages. The actions that follow then delete the Import Errors tables. After the macro executes those actions, the macro simply ends—it does *not* execute SetWarnings with the Warnings On = Yes argument in order to turn the warning messages back on. Access automatically turns the messages back on when the macro ends.

However, there's no such safety net for Access Basic functions. As you can see in the DeleteImportErrors() function, we explicitly set the messages back on before the function ends. You can probably guess why. If you don't turn the messages back on, Access won't present any confirmation dialog boxes—even after the function ends. In this situation, you could then delete objects from the Database window without confirming, delete records from the table without confirming, and so on.

Turning off the warning messages can be nice in many situations. In fact, our first recommendation in the accompanying letter was to delete the tables one at a time by pressing the [Shift][Del] key combination, which immediately deletes the highlighted table. You save a lot of time by avoiding the dialog box. However, you don't want that situation to be the norm. If you omit the *DoCmd SetWarnings True* statement from the end of an Access Basic function, you'll unintentionally turn off the warning message during interactive use.

Table A

Conditions	Actions	Action Arguments
MsgBox("Delete tables?",292) = 7	StopMacro	
	SetWarnings	Warnings On = No
	SelectObject	Object Type = Table
		Object Name = Import Errors
		In Database Window = Yes
	DoMenuItem	Menu Bar = Database
		Menu Name = Edit
		Command = Delete
	SelectObject	Object Type = Table
		Object Name = Problems
		In Database Window = Yes
	DoMenuItem	Menu Bar = Database
		Menu Name = Edit
		Command = Delete

Creating a macro to delete the tables

Ms. Barton suggests we create a macro. As you know, macros are very useful tools for automating your manual actions. However, macros aren't very good at making decisions. In terms of the problem at hand, you can create a macro that highlights and then deletes each table you want to remove. However, you can't create a macro that automatically highlights all table names that begin with the name *Import Errors*. You must include separate macro actions that highlight and delete each table.

Suppose the database contains the two tables *Import Errors - Robin Barton* and *Import Errors - Robin Barton1*. (As you may know, Access appends the base table name, *Import Errors*, to your name and then a number to avoid duplicating existing table names.) You want a macro that deletes both tables automatically.

Table A defines such a macro. The table's first column lists expressions you'd type in a macro's Conditions column, the second column lists the actions as they appear in the macro's Actions column, and the third column contains the argument settings you'd enter in the Action Arguments section.

The macro first produces a confirmation message box and then turns off the confirmation menus that would appear each time the macro tried to delete a table. Then, the macro deletes the tables. To delete each table, the macro issues a pair of actions. It first issues the *SelectObject* action to highlight a table and then runs the *DoMenuItem* action to execute the *Delete* command from the *Edit* menu.

Creating an Access Basic function to delete the tables

If you have a large number of *Import Errors* tables in your database and you expect them to appear regularly, you'd probably appreciate a utility that deletes them all at a touch of a button. Since macros can't do that, you must create an Access Basic function. The function we'll show you will scan the database window, looking for all table names that begin with *Import Errors* and will then delete them.

To create this function, first create a new module by clicking the *Module* button in the *Database* window and then clicking *New*. When the *Module* window appears, issue the *New Procedure...* command from the *Edit* menu. In the *New Procedure* dialog box, enter the *DeleteImportErrors* procedure name and click *OK*. In the window

Microsoft Access
Technical Support
 (206) 635-7050

Please include account number from label with any correspondence.

that appears, type the function's Access Basic statements:

```
Function DeleteImportErrors ()

Dim DB As Database, ASnapShot As Snapshot

If MsgBox("Delete tables?",292) = 7 Then
Exit Function
End If

Set DB = CurrentDB()
Set ASnapShot = DB.ListTables()




DoCmd SetWarnings False
Do Until ASnapShot.EOF
If ASnapShot.[Name] Like "Import Errors*"
Then
DoCmd SelectObject A_TABLE,
ASnapShot.Name, True
DoCmd DoMenuItem A_DATABASE, A_EDIT,
A_DELETE
End If
ASnapShot.MoveNext
Loop
DoCmd SetWarnings True

End Function
```

This function first displays a confirmation dialog box that asks whether you really want to delete the tables. Next, the function uses the ListTables() method to create a snapshot,

named ASnapShot, that contains all the tables in the database. The function can then search the snapshot for table names beginning with *Import Errors*, just as if the table names resided in one table.

The Do loop contains the statements that actually delete the table names. The If statement checks whether the table name stored in the snapshot's current row is an Import Errors table. If so, the next two statements highlight that table in the Database window and then delete it. The statement that follows the End If statement moves to the snapshot's next row so the function can consider the next table name for deletion. The Do loop repeats this process until it has processed every row in the snapshot.

Now, you must create a means for running the function. We suggest creating a button on a new form and calling the function from the button's On Push property. To do this, click the Forms button in the Database window and then click New. In the New Form dialog box, click BlankForm. When the Form window appears, place a new button on the form by using the Button tool  from the Toolbox. Finally, open the property sheet by clicking the Properties button  on the tool bar. Enter =DeleteImportErrors() in the On Push property. To run the function, you click the Form View button  on the tool bar in order to use the form and then click the button you just created. ♦

Subscribe to the Inside Microsoft Access Resource Disk!



Do you wish that you could experiment with the forms, reports, tables, macros, modules, and queries we regularly feature in *Inside Microsoft Access* but don't have the time or patience to create them? If so, you may want to subscribe to the Inside Microsoft Access Resource Disk. Once you subscribe, we'll send you a monthly disk loaded with all the useful tips featured in that month's *Inside Microsoft Access*. See the articles marked with the disk icon.

A six-month subscription to the Inside Microsoft Access Resource Disk costs \$29. A full one-year subscription is \$49. If you don't want to subscribe but would like the forms, reports, tables, macros, modules, and queries in a particular issue of *Inside Microsoft Access*, you can purchase a single disk for only \$9.95.

To subscribe or order a specific month's disk, just call our Customer Relations department at (800) 223-8720. Outside the US, please call (502) 491-1900.

